Advisory on Software Bill of Materials and Real-time Vulnerability Monitoring for Open-Source Software and Third-Party Dependencies

Introduction

1. The integration of Open-Source Software (OSS) in software development introduces significant cybersecurity challenges, particularly regarding vulnerabilities in third-party dependencies. Notable incidents, such as Log4j and Heartbleed, underscore these risks. On Log4j, many organisations struggled to assess system compromises due to a lack of visibility into their software components and dependencies, with delayed responses to discovered vulnerabilities. On Heartbleed, it affected the widely used OpenSSL cryptography library, leading to the theft of 4.5 million medical records from a major overseas hospital chain.

2. These dependency threats are exacerbated by extent of third-party dependencies and critical vulnerabilities found in software development projects. According to studies, there are on average 68.8[1] dependencies per project and 5.1[2] critical vulnerabilities in an application. If developers are unaware of the full composition of their applications, the risks of cybersecurity breaches are significant. In the light of such trends, there is an impetus for developers to easily identify and address OSS dependencies to mitigate cybersecurity risks.

Intended audience of advisory

3. The advisory is intended for all software developers, especially those who incorporate OSS and third-party dependencies into their projects. While many developers are aware of cybersecurity risks, they may not have the resources and guidance to enforce cybersecurity during software development and implementation. To aid developers, the advisory offers guidance on a sustainable and automated approach to vulnerability management through Software Bill of Materials (SBOM) and real-time vulnerability monitoring.

[1] https://chenbihuan.github.io/paper/icsme20-wang-lib-empirical.pdf
[2] https://www.helpnetsecurity.com/2022/10/04/when-transparency-is-also-obscurity-open-source-security/

Value proposition of SBOM and real-time monitoring of vulnerabilities

4. The traditional way of manually managing OSS dependencies is inefficient and prone to errors. Furthermore, developers must sift through complex codebases to identify and fix vulnerable software components.

5. **Generating Software Bill of Materials (SBOM) ensures developers are not using known vulnerable dependencies and provide them full visibility into software components**. SBOM provides a structured and formal record of components used to build software. It equips organisations with a clear view of their software environment, ensuring that vulnerabilities can be managed more effectively. Through the integration of SBOM tools into software development workflows, developers can automatically track each component from the start, reducing manual effort and human error. It enables them to significantly reduce technical debt by identifying outdated or vulnerable components much earlier, in turn decreasing future remediation workloads.

6. **SBOM also improves response times by allowing developers to quickly identify and fix vulnerable components and collaborate across the organisation for holistic vulnerability management.** This streamlined process not only minimises complexity but also fosters collaboration among developers and cybersecurity professionals, allowing cybersecurity risks to be addressed proactively without stifling innovation. If SBOM is integrated into CI/CD pipelines, it allows real-time monitoring of new vulnerabilities through automation of SBOM generation, signing and alerts. The SBOM can also be used to foster collaboration across teams, including SecOps, Incident Response (IR) and development teams for holistic vulnerability management and improved response times.

Three step approach to managing vulnerabilities through SBOMs

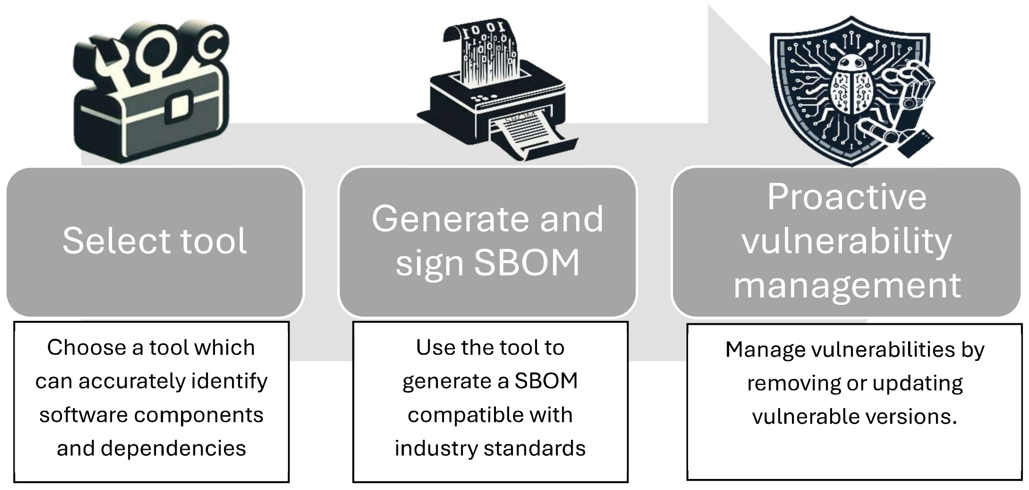
Select tool
Generate and sign SBOM
Proactive vulnerability management
Choose a tool which can accurately identify software components and dependencies
Use the tool to generate a SBOM compatible with industry standards
Manage vulnerabilities by removing or updating vulnerable versions.

Figure 1. Three step approach to managing vulnerabilities through SBOMs

7. The three-step approach is as follows[3]:

- **Select tool:** The chosen tool should accurately identify and list components as well as direct[4] and indirect[5] dependencies of the software. The tool should also integrate seamlessly with continuous integration/continuous deployment (CI/CD) pipelines such as GitHub Actions, GitLab CI/CD[6] or equivalent software.

- **Generate and sign the SBOM:** The tool should be used to generate an SBOM that complies with industry standards such as CycloneDX or SPDX[7]. Signing the SBOM after its generation ensures authenticity and provides assurance that it originates from a trusted source. Developers should leverage on available tools[8] to publish signed records into transparency logs, enhancing trust and verifiability through immutable records of signing events.

- **Proactive vulnerability management:** The generated SBOM should be published to a secure repository and automatically ingested by tools like OWASP Dependency-Track[9] for continuous vulnerability monitoring and N-day vulnerability identification.

8. As the software development environment varies for each system, developers should also take note of the following practical considerations:

- **The SBOM is only as comprehensive as the manifest files generated.** If the codebase includes obscure or less common programming languages, some dependencies may not be detected;

- For **SaaS and closed-source software**, developers should request the SBOMs from their third-party providers as these SBOMs provide the most comprehensive coverage of the software components and dependencies. If developers are unable to obtain SBOMs from their third-party providers, the selected SBOM generation tool need to be able to perform supplementary checks in the respective environments. In particular, SaaS software could use

[3] The approach is adapted from existing best practices found in sources 1,3 to 6 of bibliography.
[4] Direct refers to software components that are explicitly required by the code.
[5] Indirect refers to software components required by the direct dependencies.
[6] GitHub Actions and GitLab CI/CD are CI/CD tools to automate workflows like building, testing, and deploying code.
[7] These are industry standards for SBOM creation, distribution, and consumption, enabling interoperability and integration into existing workflows.
[8] An example of such a tool is Sigstore, which provides signing, verification, and provenance checks to secure Open-Source Software distribution.
[9] OWASP Dependency-Track tracks and identify software vulnerabilities through SBOM analysis.

runtime SBOMs[10], capturing dynamically loaded or run-time injected components during execution. For closed-source software, binary-based SBOM tools[11] could be used to create an inventory of the software components used. Such tools inspect the compiled binary code, which is the final product of the software. Once developers discover vulnerabilities through the SBOMs, they need to inform their third-party providers to remediate them;

- **Developers need to verify identified vulnerabilities for exploitability as the vulnerabilities may not be relevant in their software development environments**. Without appropriate verification, there could be a high volume of vulnerabilities surfaced through the SBOM and developers risk being overwhelmed with false positives and time spent on subsequent remediation. Developers should start the verification process by using industry filters such as CISA's Known Exploited Vulnerabilities (KEV)[12] that only alerts on CVEs that are actively exploited. Once the vulnerabilities are filtered, vulnerabilities can be assessed for the likelihood of being exploited through the Exploit Prediction Scoring System (EPSS)[13]. After identifying vulnerabilities that are likely to be exploited, they can be prioritised for remediation using Common Vulnerability Scoring Systems (CVSS), which rates the severity of vulnerabilities.

<u>Automating capabilities in code repositories platforms to manage OSS vulnerabilities</u>

9. Both commercial and Open-Source Software projects commonly rely on open-source dependencies, typically hosted on code repository platforms such as GitHub and GitLab. As central hubs for OSS development, GitHub and GitLab are used by developers who collaborate on projects of varying scope and complexity. Given the widespread use of such code repository platforms, managing vulnerabilities are even more critical for maintaining a secure software ecosystem. Such environments provide automated workflow functionalities on managing vulnerabilities, with automation enabling a seamless integration of security practices into the development process.

[10] Runtime SBOMs requires the system to be analysed when running. Some detailed information may be available only after the system has been run for a period of time until the complete functionality has been exercised. Example of runtime SBOM tools are *Anchor Syft and Slim.AI*.
[11] Example of binary-based SBOM tools are *Black Duck Binary Analysis and Tern*.
[12] CISA's Known Exploited Vulnerabilities (KEV) is a catalogue of actively exploited vulnerabilities to help prioritise security remediation.
[13] Exploit Prediction Scoring System (EPSS) is a predictive model that scores the probability of a software vulnerability being exploited, aiding in prioritising security responses.

10. **Developers should use tools such as GitHub Actions and GitLab CI/CD that allow for the automated creation and vulnerability checking of SBOMs.** While SBOM signing is not a native feature, external tools [14] can be integrated into the workflow. Refer to Annex A for the full script that automates these actions for GitHub Actions and a similar workflow can be implemented for GitLab CI/CD (Annex B).

11. **Developers should either remove vulnerable components if the functionalities provided through these components are not crucial or update these components to non-vulnerable versions.** Developers should thoroughly test their applications to verify the application works as intended and update the SBOM documentation to record which components were removed and updated. This approach enhances the cybersecurity of the software and protects users from known exploits.

12. **Developers should publish SBOM, its signature and certificate alongside the digital files.** This allows downstream users in the software development ecosystem to easily access and verify the SBOM, ensuring that they are working with a secure and authentic version of the software. Users can also leverage the SBOM to continuously monitor for new vulnerabilities.

Real-time monitoring of vulnerabilities through OWASP Dependency Track

13. OWASP Dependency Track (DT) provides real-time vulnerability monitoring capabilities through SBOM ingestion and continual checking against current threat intelligence. OWASP Dependency Track (DT) goes beyond basic scanning by incorporating the Exploit Prediction Scoring System (EPSS), allowing developers to prioritise vulnerabilities based on their likelihood of exploitation. Refer to Annex C for more details on the deployment of OWASP DT.

14. **Developers should integrate the OWASP DT tool into CI/CD pipelines for real-time monitoring, consistent automation of SBOM generation and signing, and alerts for new vulnerabilities.** Developers should securely store signed SBOM into centralised repositories to support collaboration across teams, including SecOps, Incident Response (IR) and development teams. In addition, developers need to establish governance policies for SBOM storage, access control and lifecycle management in collaboration with their Chief Information Security Officers (CISOs).

[14] An example is Cosign, which is a tool to sign, verify and attest software artifacts securely.

Conclusion

15. SBOMs and real-time monitoring of vulnerabilities provide developers a sustainable and automated approach to address risks posed by Open-Source Software (OSS) and third-party software components, in turn enhancing the cybersecurity posture of the software supply chain. Such an approach allows developers and system owners to have visibility on software components and dependencies and improve response times to address vulnerabilities.

Acknowledgement

16. This advisory was jointly developed by the Cyber Security Agency of Singapore and OWASP Foundation.

Disclaimer

17. The information and advice contained in this document is provided "as is" without any warranties or guarantees. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favouring by the Cyber Security Agency of Singapore, OWASP Foundation or GitHub. This document shall not be used for advertising or product endorsement purposes.

List of References

S/N	Document	Source	Year of Publication
1	The Minimum Elements for a Software Bill of Materials (SBOM)	NTIA	2021
2	Addressing Cybersecurity Challenges in Open-Source Software	OpenSSF	2022
3	Guidance on Introduction of Software Bill of Materials (SBOM) for Software Management	METI	2023
4	Recommendations for Software Bill of Materials (SBOM) Management	NSA	2024
5	Documentation on OWASP Dependency-Track	OWASP	2024, v4.11
6	CycloneDX Authoritative Guide to SBOM	OWASP	2024, 2nd edition

Annex A – Sample workflow. yaml script for GitHub Actions

On every code commit to the repository, this workflow script will create an SBOM by importing and using an open-source tool[15]. The components, its version numbers and ecosystem will also be displayed on the console. This SBOM is signed[16] as an arbitrary binary large object and a signature with certificate is generated. Finally, the SBOM is ingested[17] to check the components for vulnerabilities. The components, vulnerable version, fixed version, ecosystem, security advisory and criticality of vulnerability will also be displayed on the console.

To start the workflow, create a .yml file in ./github/workflows/, for example, SBOMgen-VA.yml, with the sample script (shown below) that provides the necessary actions for the workflow to run.

```yaml
# Workflow name that appears in the GitHub Actions UI
name: Generate, Sign, and Commit SBOM

# Trigger the workflow on a push to the 'main' branch
on:
  push:
    branches:
      - main
  workflow_dispatch: # Allow manual triggering of the workflow

# Define jobs in the workflow
jobs:
  # Job for generating and signing the SBOM
  generate-and-sign-sbom:
    runs-on: ubuntu-latest  # Specifies the runner environment
    permissions:  # Sets permissions for the GitHub token
      packages: write
      id-token: write
      contents: write
    steps:
      - name: Checkout code  # Checkout the repository code
        uses: actions/checkout@v2
```

[15] Syft is an example of a tool that can identify and list both direct and indirect dependencies and integrate seamlessly with GitHub Actions. Alternatives include but are not limited to Tern, CycloneDX Generator and SPDX SBOM Generator.
[16] Cosign is an example of a tool that can sign an arbitrary binary large object. Alternatives include but are not limited to Notary, OpenSSL, Sigstore Rekor.
[17] Grype is an example of a tool that can ingest SBOMs in different formats and output additional information for the detected vulnerability like the relevant CVE or security advisory entry with a criticality rating. Alternatives include but are not limited to Trivy and Clair.

```

    - name: Install Syft  # Install Syft for generating SBOM
      run: |
        # Download and install Syft script from the official source
        curl -sSfL https://raw.githubusercontent.com/anchore/syft/main/install.sh | sh -s -- -b /usr/local/bin
        #Verify Syft installation
        syft -v
    - name: Generate SBOM in CycloneDX format  # Generate the SBOM
      run: |
        # Generate SBOM from the repository content and save it in CycloneDX JSON format
        syft . -o cyclonedx-json=log4shell-vulnerable-app-CDX.json

    - name: Upload SBOM as an artifact  # Upload SBOM as a build artifact
      uses: actions/upload-artifact@v2
      with:
        name: SBOM
        path: log4shell-vulnerable-app-CDX.json

    - name: Install Cosign  # Install Cosign for signing artifacts
      run: |
        # Download Cosign, make it executable, and move to local bin
        COSIGN_VERSION="v2.2.3"
        wget https://github.com/sigstore/cosign/releases/download/${COSIGN_VERSION}/cosign-linux-amd64 -O cosign
        chmod +x cosign
        sudo mv cosign /usr/local/bin/

    - name: Sign the SBOM and generate certificate  # Sign the SBOM and generate a verification certificate
      env:
        COSIGN_EXPERIMENTAL: "1"
      run: |
        # Use Cosign to sign the SBOM, specify OIDC issuer, and output both signature and certificate
        cosign sign-blob --oidc-issuer="https://token.actions.githubusercontent.com" \
        --yes \
        --output-signature log4shell-vulnerable-app-CDX.json.sig \
        --output-certificate log4shell-vulnerable-app-CDX.pem \
        log4shell-vulnerable-app-CDX.json

    - name: Commit SBOM, Signature, and Certificate to repository  # Commit the SBOM, signature, and certificate to the repo
      run: |
          # Configure git with user credentials, add files, and commit them to the repository
          git config --local user.email "<your_email@example.com>"
          git config --local user.name "<Your GitHub Username>"
          git add log4shell-vulnerable-app-CDX.json log4shell-vulnerable-app-CDX.json.sig log4shell-vulnerable-app-CDX.pem
          git commit -m "Add and Sign SBOM for log4shell-vulnerable-app"
          git push

  # Additional job for vulnerability analysis
  vulnerability-analysis:
    needs: generate-and-sign-sbom  # Dependency on the first job
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code  # Checkout the repository code again for analysis
        uses: actions/checkout@v2

      - name: Download SBOM artifact  # Download the previously uploaded SBOM artifact
        uses: actions/download-artifact@v2
        with:
          name: SBOM

      - name: Install Grype  # Install Grype for vulnerability scanning
        run: |
          # Download and install Grype script from the official source
          curl -sSfL https://raw.githubusercontent.com/anchore/grype/main/install.sh | sh -s -- -b /usr/local/bin

      - name: Run vulnerability analysis  # Perform vulnerability analysis on the SBOM
        run: |
          # Use Grype to scan the SBOM and output results in a table format
          grype sbom:log4shell-vulnerable-app-CDX.json -o table

```

Annex B – Sample workflow. yaml script for GitLab CI/CD

To start the workflow, create a .gitlab-ci.yml file at the root of your repository with the following content with the sample script (shown below) that provides the necessary actions for the workflow to run:

```
stages:
  - generate_and_sign_sbom
  - vulnerability_analysis

generate-and-sign-sbom:
  stage: generate_and_sign_sbom
  image: ubuntu:latest
  script:
    - apt-get update && apt-get install -y curl wget sudo git

    # Install Syft
    - curl -sSfL https://raw.githubusercontent.com/anchore/syft/main/install.sh | sh -s -- -b /usr/local/bin
    # Verify Syft installation
    - syft -v
    # Generate SBOM in CycloneDX format
    - syft . -o cyclonedx-json=log4shell-vulnerable-app-CDX.json

    # Upload SBOM artifact
    - echo "Uploading SBOM artifact"
    - mv log4shell-vulnerable-app-CDX.json $CI_PROJECT_DIR/

    # Install Cosign
    - COSIGN_VERSION="v2.2.3"
    - wget https://github.com/sigstore/cosign/releases/download/${COSIGN_VERSION}/cosign-linux-amd64 -O cosign
    - chmod +x cosign
    - sudo mv cosign /usr/local/bin/

    # Sign the SBOM and generate certificate
    - export COSIGN_EXPERIMENTAL=1
    - cosign sign-blob --yes --output-signature log4shell-vulnerable-app-CDX.json.sig --output-certificate log4shell-vulnerable-app-CDX.pem log4shell-vulnerable-app-CDX.json

    # Commit SBOM, Signature, and Certificate to repository
    - git config --global user.email "<your_email@example.com>"
    - git config --global user.name "<Your GitLab Username>"
    - git add log4shell-vulnerable-app-CDX.json log4shell-vulnerable-app-CDX.json.sig log4shell-vulnerable-app-CDX.pem
    - git commit -m "Add and Sign SBOM for log4shell-vulnerable-app"
    - git push origin $CI_COMMIT_BRANCH
  artifacts:
    paths:
      - log4shell-vulnerable-app-CDX.json
      - log4shell-vulnerable-app-CDX.json.sig
      - log4shell-vulnerable-app-CDX.pem

vulnerability-analysis:
  stage: vulnerability_analysis
  image: ubuntu:latest
  script:
    - apt-get update && apt-get install -y curl wget sudo git

    # Download SBOM artifact
    - cp $CI_PROJECT_DIR/log4shell-vulnerable-app-CDX.json ./

    # Install Grype
    - curl -sSfL https://raw.githubusercontent.com/anchore/grype/main/install.sh | sh -s -- -b /usr/local/bin

    # Run vulnerability analysis
    - grype sbom:log4shell-vulnerable-app-CDX.json -o table
  dependencies:
    - generate-and-sign-sbom
```

Annex C – Deploying and using Dependency-Track

Deploying DT using the provided Docker image from dependencytrack.org is the most convenient way to get started. The following commands are the easiest way to deploy DT, assuming a Docker installation has been setup properly:

```
# Downloads the latest Docker Compose file
curl -LO https://dependencytrack.org/docker-compose.yml

# Starts the stack using Docker Compose
docker-compose up -d
```

After deploying DT and logging into DT at http://localhost:8080/ with the default credentials (username: admin, password: admin), create a project with the appropriate details. Select the created project and upload the SBOM in the 'components' tab. After completion of ingestion and vulnerability database updating, DT will automatically and continuously scan for vulnerabilities and display known vulnerabilities in the 'audit vulnerabilities' tab. This process can be repeated for other projects and a global view of all components in all projects is automatically collated in the 'components' tab in the sidebar. Developers are advised to check the known vulnerable components and prioritise remediation. Refer to DT's official documentation at https://doc.dependencytrack.org/getting-started/ for more details on getting started.

For more advanced users, DT includes a built-in API that integrates seamlessly with Continuous Integration/Continuous Delivery (CI/CD) pipelines. This feature allows developers to automate ingestion of SBOMs and vulnerability checks as part of their Software Development Lifecycle (SDLC) workflow, enabling a more robust security posture. Additionally, the workflow script can include an optional step to publish the generated SBOM directly to DT for continuous monitoring and analysis, ensuring that vulnerabilities are tracked and managed throughout the software's lifecycle.